圆方树

算法简介

一种可以将图的简短路径操作转化为树上路径操作的算法。

算法实现

首先给出圆方树相关定义：

对无向图求双连通分量，然后给每个双连通分量建一个新点。

特别的，这里点双连通分量的定义是不存在割点的最大子图，然后只有一个点的图需要自行特殊处理。

每个双连通分量代表的方点向原图中属于该点双连通分量的点连一条边，得到一棵树，称为圆方树。

同时称双连通分量代表的点为方点，原图中的点为圆点。

具体实现稍微修改一下求点双连通分量的代码即可，记得由于点双连通分量最多有 $O(n)$ 个要开双倍数组。

```
vector<int> g[MAXN<<1];
int node_cnt;
int low[MAXN],dfs_id[MAXN],dfs_t,bcc_id[MAXN],bcc_cnt;
vector<int> bcc[MAXN];
stack<pair<int,int> >Stack;
bool iscut[MAXN];
void dfs(int u,int fa){
    low[u]=dfs_id[u]=++dfs_t;
    blk_sz++;
    int child=0;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==fa)continue;
        if(!dfs_id[v]){
            Stack.push(make_pair(u,v));
            dfs(v,u);
            low[u]=min(low[u],low[v]);
            if(low[v]>=dfs_id[u]){
                iscut[u]=true;
                pair<int,int> temp;
                bcc[++bcc_cnt].clear();
                while(true){
                    temp=Stack.top();Stack.pop();
                    if(bcc_id[temp.first]!=bcc_cnt){
                        bcc_id[temp.first]=bcc_cnt;
                        bcc[bcc_cnt].push_back(temp.first);
                    }
```

```
                    if(bcc_id[temp.second]!=bcc_cnt){
                        bcc_id[temp.second]=bcc_cnt;
                        bcc[bcc_cnt].push_back(temp.second);
                    }
                    if(temp.first==u&&temp.second==v)
                    break;
                }
                node_cnt++;//就加了几行
                for(int node_id:bcc[bcc_cnt]){
                    g[node_cnt].push_back(node_id);
                    g[node_id].push_back(node_cnt);
                }
            }
            child++;
        }
        else if(dfs_id[v]<dfs_id[u]){
            Stack.push(make_pair(u,v));
            low[u]=min(low[u],dfs_id[v]);
        }
    }
    if(u==fa&&child<2)
    iscut[u]=false;
}
void find_bcc(int n){
    mem(dfs_id,0);
    mem(iscut,0);
    mem(bcc_id,0);
    dfs_t=bcc_cnt=0;
    node_cnt=n;
    _rep(i,1,n){
        if(!dfs_id[i])
        dfs(i,i);
    }
}
```

算法例题

例题一

洛谷p4630

题意

给定一个图，不保证连通。求有多少三元组 (s,c,f) 满足 s,c,f 互异且存在一条从 s 出发经过 c 到达 f 的简单路径。

题解

首先给定点双连通分量的一个性质，任取一个点双连通分量中互异的三个点 (s,c,f) 一定存在一条从 s 出发经过 c 到达 f 的简单路径。

该性质由来见 证明

然后又可以得出结论任取两个圆点 s,f 则满足条件的 c 正好是与 s,f 在圆方树的路径经过的方点相邻的圆点构成的集合(注意减去 s,f 本身)。

不妨设每个方点的点权为该点双连通分量的大小，每个圆点权值为 -1。则不难发现 s,f 对应的合法的 c 数量就是 s,f 在圆方树路径上的点权和。

考虑树形 dp 求解经过每个点的路径个数计算贡献即可，时间复杂度 $O(n+m)$

```
const int MAXN=1e5+5,MAXM=2e5+5;
struct Edge{
    int to,next;
}edge[MAXM<<1];
int head[MAXN],edge_cnt;
void Insert(int u,int v){
    edge[++edge_cnt]=Edge{v,head[u]};
    head[u]=edge_cnt;
}
vector<int> g[MAXN<<1];
int node_cnt,blk_sz,val[MAXN<<1];
int low[MAXN],dfs_id[MAXN],dfs_t,bcc_id[MAXN],bcc_cnt;
vector<int> bcc[MAXN];
stack<pair<int,int> >Stack;
bool iscut[MAXN];
void dfs(int u,int fa){
    low[u]=dfs_id[u]=++dfs_t;
    blk_sz++;
    int child=0;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==fa)continue;
        if(!dfs_id[v]){
            Stack.push(make_pair(u,v));
            dfs(v,u);
            low[u]=min(low[u],low[v]);
            if(low[v]>=dfs_id[u]){
                iscut[u]=true;
                pair<int,int> temp;
                bcc[++bcc_cnt].clear();
                while(true){
                    temp=Stack.top();Stack.pop();
                    if(bcc_id[temp.first]!=bcc_cnt){
                        bcc_id[temp.first]=bcc_cnt;
                        bcc[bcc_cnt].push_back(temp.first);
```

```
                    }
                    if(bcc_id[temp.second]!=bcc_cnt){
                        bcc_id[temp.second]=bcc_cnt;
                        bcc[bcc_cnt].push_back(temp.second);
                    }
                    if(temp.first==u&&temp.second==v)
                    break;
                }
                val[++node_cnt]=bcc[bcc_cnt].size();
                for(int node_id:bcc[bcc_cnt]){
                    g[node_cnt].push_back(node_id);
                    g[node_id].push_back(node_cnt);
                }
            }
            child++;
        }
        else if(dfs_id[v]<dfs_id[u]){
            Stack.push(make_pair(u,v));
            low[u]=min(low[u],dfs_id[v]);
        }
    }
    if(u==fa&&child<2)
    iscut[u]=false;
}
int sz[MAXN<<1];
LL ans;
void dfs2(int u,int fa){
    if(val[u]==-1)sz[u]=1;
    for(int v:g[u]){
        if(v==fa)continue;
        dfs2(v,u);
        ans+=2LL*sz[u]*sz[v]*val[u];
        sz[u]+=sz[v];
    }
    ans+=2LL*sz[u]*(blk_sz-sz[u])*val[u];
}
void solve(int rt){
    blk_sz=0;
    dfs(rt,rt);
    dfs2(rt,rt);
}
void find_bcc(int n){
    mem(dfs_id,0);
    mem(iscut,0);
    mem(bcc_id,0);
    dfs_t=bcc_cnt=0;
    node_cnt=n;
    _rep(i,1,n)
    val[i]=-1;
    _rep(i,1,n){
```

```
        if(!dfs_id[i])
        solve(i);
    }
}
int main(){
    int n=read_int(),m=read_int();
    while(m--){
        int u=read_int(),v=read_int();
        Insert(u,v);
        Insert(v,u);
    }
    find_bcc(n);
    enter(ans);
    return 0;
}
```

例题二

CF487E

题意

给定一个点权图，维护两种操作：

1. 修改某个点的点权
2. 询问从 u 到 v 所有简短路径上的点的点权最小值

题解

同样建圆方树，然后考虑每个方点用一个 set 维护周围圆点的点权最小值，然后树剖处理询问，时间复杂度 $O\left(n\log^2 n\right)$

但是注意到一个割点可能属于 $O(n)$ 个双连通分量，不能直接更新所有相邻方点。于是考虑每个方点维护圆方树中所有儿子结点的点权最小值。

这样更新一个圆点点权时最多更新一个方点，于是更新操作时间复杂度 $O(n\log n)$

但注意这样转化后，对查询操作，如果查询两圆点的 LCA 为方点，则需要补上方点的父结点贡献。

```
const int MAXN=1e5+5,MAXM=1e5+5;
const int MAXN2=MAXN<<1,MAXM2=MAXM<<1,inf=1e9;
struct Edge{
    int to,next;
};
namespace SegTree{
    int lef[MAXN2<<2],rig[MAXN2<<2],s[MAXN2<<2];
    int *a;
```

```
    void push_up(int k){
        s[k]=min(s[k<<1],s[k<<1|1]);
    }
    void build(int k,int L,int R){
        lef[k]=L,rig[k]=R;
        int M=L+R>>1;
        if(L==R){
            s[k]=a[M];
            return;
        }
        build(k<<1,L,M);
        build(k<<1|1,M+1,R);
        push_up(k);
    }
    void init(int *_a,int n){
        a=_a;
        build(1,1,n);
    }
    void update(int k,int pos,int v){
        if(lef[k]==rig[k]){
            s[k]=v;
            return;
        }
        int mid=lef[k]+rig[k]>>1;
        if(pos<=mid)
        update(k<<1,pos,v);
        else
        update(k<<1|1,pos,v);
        push_up(k);
    }
    int query(int k,int L,int R){
        if(L<=lef[k]&&rig[k]<=R)
        return s[k];
        int mid=lef[k]+rig[k]>>1;
        if(mid>=R)
        return query(k<<1,L,R);
        else if(mid<L)
        return query(k<<1|1,L,R);
        else
        return min(query(k<<1,L,R),query(k<<1|1,L,R));
    }
}
namespace Tree{
    Edge edge[MAXM2<<1];
    int n,head[MAXN2],edge_cnt,val[MAXN2];
    void Insert(int u,int v){
        edge[++edge_cnt]=Edge{v,head[u]};
        head[u]=edge_cnt;
    }
    int d[MAXN2],sz[MAXN2],f[MAXN2],dfs_id[MAXN2],dfs_t;
```

```
    int h_son[MAXN2],mson[MAXN2],p[MAXN2],dfs_w[MAXN2];
    multiset<int> s[MAXN2];
    void dfs_1(int u,int fa,int depth){
        sz[u]=1;f[u]=fa;d[u]=depth;mson[u]=0;
        for(int i=head[u];i;i=edge[i].next){
            int v=edge[i].to;
            if(v==fa)
            continue;
            dfs_1(v,u,depth+1);
            sz[u]+=sz[v];
            if(sz[v]>mson[u]){
                h_son[u]=v;
                mson[u]=sz[v];
            }
        }
        if(u>n){
            for(int i=head[u];i;i=edge[i].next){
                int v=edge[i].to;
                if(v==fa)continue;
                s[u].insert(val[v]);
            }
            val[u]=*s[u].begin();
        }
    }
    void dfs_2(int u,int top){
        dfs_id[u]=++dfs_t;p[u]=top;dfs_w[dfs_t]=val[u];
        if(mson[u])
        dfs_2(h_son[u],top);
        for(int i=head[u];i;i=edge[i].next){
            int v=edge[i].to;
            if(v==f[u]||v==h_son[u])
            continue;
            dfs_2(v,v);
        }
    }
    void build(int _n){
        n=_n;
        dfs_1(1,0,0);
        dfs_2(1,1);
        SegTree::init(dfs_w,n<<1);
    }
    void update(int u,int w){
        int t=val[u];
        val[u]=w;
        SegTree::update(1,dfs_id[u],val[u]);
        if(f[u]){
            s[f[u]].erase(t);
            s[f[u]].insert(val[u]);
            val[f[u]]=*s[f[u]].begin();
            SegTree::update(1,dfs_id[f[u]],val[f[u]]);
        }
```

```
    }
    int query(int u,int v){
        int ans=inf;
        while(p[u]!=p[v]){
            if(d[p[u]]<d[p[v]])
            swap(u,v);
            ans=min(ans,SegTree::query(1,dfs_id[p[u]],dfs_id[u]));
            u=f[p[u]];
        }
        if(d[u]>d[v])
        swap(u,v);
        ans=min(ans,SegTree::query(1,dfs_id[u],dfs_id[v]));
        if(u>n)
        ans=min(ans,val[f[u]]);
        return ans;
    }
}
Edge edge[MAXM<<1];
int head[MAXN],edge_cnt;
void Insert(int u,int v){
    edge[++edge_cnt]=Edge{v,head[u]};
    head[u]=edge_cnt;
}
int node_cnt;
int low[MAXN],dfs_id[MAXN],dfs_t,bcc_id[MAXN],bcc_cnt;
vector<int> bcc[MAXN];
stack<pair<int,int> >Stack;
void dfs(int u,int fa){
    low[u]=dfs_id[u]=++dfs_t;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==fa)continue;
        if(!dfs_id[v]){
            Stack.push(make_pair(u,v));
            dfs(v,u);
            low[u]=min(low[u],low[v]);
            if(low[v]>=dfs_id[u]){
                pair<int,int> temp;
                bcc[++bcc_cnt].clear();
                while(true){
                    temp=Stack.top();Stack.pop();
                    if(bcc_id[temp.first]!=bcc_cnt){
                        bcc_id[temp.first]=bcc_cnt;
                        bcc[bcc_cnt].push_back(temp.first);
                    }
                    if(bcc_id[temp.second]!=bcc_cnt){
                        bcc_id[temp.second]=bcc_cnt;
                        bcc[bcc_cnt].push_back(temp.second);
                    }
                    if(temp.first==u&&temp.second==v)
```

```
                    break;
                }
                node_cnt++;
                for(int node_id:bcc[bcc_cnt]){
                    Tree::Insert(node_cnt,node_id);
                    Tree::Insert(node_id,node_cnt);
                }
            }
        }
        else if(dfs_id[v]<dfs_id[u]){
            Stack.push(make_pair(u,v));
            low[u]=min(low[u],dfs_id[v]);
        }
    }
}
int main(){
    int n=read_int(),m=read_int(),q=read_int();
    _rep(i,1,n)
    Tree::val[i]=read_int();
    while(m--){
        int u=read_int(),v=read_int();
        Insert(u,v);
        Insert(v,u);
    }
    node_cnt=n;
    dfs(1,1);
    Tree::build(n);
    while(q--){
        char opt=get_char();
        int u=read_int(),v=read_int();
        if(opt=='A')
        enter(Tree::query(u,v));
        else
        Tree::update(u,v);
    }
    return 0;
}
```

例题三

洛谷p4606

题意

给定一个连通图。接下来多次询问，每次询问一个点集 S□求有多少点 c 满足 $c\not\in S$ 且删去 c 导致 S 中的点不全部属于同一个连通分量。

题解

建圆方树，易知如果删除点 c 导致 u,v 之间在原图上不可达等价于删除点 c 导致 u,v 之间在圆方树上不可达。

于是可以求 S 中的点在圆方树上构成的虚树，易知答案为虚树中圆点数减去 S 的点数。

当然没有必要真的建立虚树，可以将圆点的点权转化为圆点到父结点的边权，然后求虚树的边权和，最后如果所有点的 LCA 是圆点，则答案加一。

至于求虚树的边权和是经典操作，只需要将关键点按 dfs 排序后依次求相邻两点距离和再除以 2 即可。时间复杂度 $O(n\log n)$

```
const int MAXN=1e5+5,MAXM=2e5+5,MAXN2=MAXN<<1;
struct Edge{
    int to,next;
};
namespace Tree{
    Edge edge[MAXN2<<1];
    int n,head[MAXN2],edge_cnt;
    void Insert(int u,int v){
        edge[++edge_cnt]=Edge{v,head[u]};
        head[u]=edge_cnt;
    }
    int d[MAXN2],sz[MAXN2],f[MAXN2],dfs_id[MAXN2],dfs_t;
    int h_son[MAXN2],mson[MAXN2],p[MAXN2],dis[MAXN2];
    void clear(int n){
        n<<=1;
        dfs_t=edge_cnt=0;
        _rep(i,1,n)
        head[i]=0;
    }
    void dfs_1(int u,int fa,int depth){
        sz[u]=1;f[u]=fa;d[u]=depth;mson[u]=0;
        dis[u]=dis[fa]+(u<=n);
        for(int i=head[u];i;i=edge[i].next){
            int v=edge[i].to;
            if(v==fa)
            continue;
            dfs_1(v,u,depth+1);
            sz[u]+=sz[v];
            if(sz[v]>mson[u]){
                h_son[u]=v;
                mson[u]=sz[v];
            }
        }
    }
    void dfs_2(int u,int top){
        dfs_id[u]=++dfs_t;p[u]=top;
```

```
        if(mson[u])
        dfs_2(h_son[u],top);
        for(int i=head[u];i;i=edge[i].next){
            int v=edge[i].to;
            if(v==f[u]||v==h_son[u])
            continue;
            dfs_2(v,v);
        }
    }
    void build(int _n){
        n=_n;
        dfs_1(1,0,0);
        dfs_2(1,1);
    }
    int lca(int u,int v){
        while(p[u]!=p[v]){
            if(d[p[u]]<d[p[v]])
            swap(u,v);
            u=f[p[u]];
        }
        return d[u]<d[v]?u:v;
    }
    int query(int u,int v){
        return dis[u]+dis[v]-2*dis[lca(u,v)];
    }
}
Edge edge[MAXM<<1];
int head[MAXN],edge_cnt;
void Insert(int u,int v){
    edge[++edge_cnt]=Edge{v,head[u]};
    head[u]=edge_cnt;
}
int node_cnt;
int low[MAXN],dfs_id[MAXN],dfs_t,bcc_id[MAXN],bcc_cnt;
vector<int> bcc[MAXN];
stack<pair<int,int> >Stack;
void dfs(int u,int fa){
    low[u]=dfs_id[u]=++dfs_t;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==fa)continue;
        if(!dfs_id[v]){
            Stack.push(make_pair(u,v));
            dfs(v,u);
            low[u]=min(low[u],low[v]);
            if(low[v]>=dfs_id[u]){
                pair<int,int> temp;
                bcc[++bcc_cnt].clear();
                while(true){
                    temp=Stack.top();Stack.pop();
                    if(bcc_id[temp.first]!=bcc_cnt){
```

```
                        bcc_id[temp.first]=bcc_cnt;
                        bcc[bcc_cnt].push_back(temp.first);
                    }
                    if(bcc_id[temp.second]!=bcc_cnt){
                        bcc_id[temp.second]=bcc_cnt;
                        bcc[bcc_cnt].push_back(temp.second);
                    }
                    if(temp.first==u&&temp.second==v)
                    break;
                }
                node_cnt++;
                for(int node_id:bcc[bcc_cnt]){
                    Tree::Insert(node_cnt,node_id);
                    Tree::Insert(node_id,node_cnt);
                }
            }
        }
        else if(dfs_id[v]<dfs_id[u]){
            Stack.push(make_pair(u,v));
            low[u]=min(low[u],dfs_id[v]);
        }
    }
}
int node[MAXN];
bool cmp(int a,int b){
    return Tree::dfs_id[a]<Tree::dfs_id[b];
}
void solve(){
    int n=read_int(),m=read_int();
    _rep(i,1,n)
    head[i]=dfs_id[i]=bcc_id[i]=0;
    bcc_cnt=dfs_t=edge_cnt=0;
    Tree::clear(n);
    while(m--){
        int u=read_int(),v=read_int();
        Insert(u,v);
        Insert(v,u);
    }
    node_cnt=n;
    dfs(1,1);
    Tree::build(n);
    int q=read_int();
    while(q--){
        int s=read_int();
        _for(i,0,s)
        node[i]=read_int();
        sort(node,node+s,cmp);
        int ans=0;
        _for(i,0,s)
        ans+=Tree::query(node[i],node[(i+1)%s]);
```

```
        ans/=2;
        if(Tree::lca(node[0],node[s-1])<=n)
        ans++;
        enter(ans-s);
    }
}
int main(){
    int T=read_int();
    while(T--){
        solve();
    }
    return 0;
}
```